PT-Scotch: A tool for efficient parallel graph ordering

Cédric Chevalier, François Pellegrini

▶ **To cite this version:**

Cédric Chevalier, François Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. 4th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'06), Sep 2006, Rennes, France. hal-00410427

HAL Id: hal-00410427
https://hal.science/hal-00410427

Submitted on 20 Aug 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PT-Scotch: A tool for efficient parallel graph ordering

Cédric Chevalier and François Pellegrini

I. Introduction

Graph partitioning is an ubiquitous technique which has applications in many fields of computer science and engineering. It is mostly used to help solving domain-dependent optimization problems modeled in terms of weighted or unweighted graphs, where finding good solutions amounts to computing, eventually recursively in a divide-and-conquer framework, small vertex or edge cuts that balance evenly the weights of the graph parts.

Because there always exists large problem graphs which cannot fit in the memory of sequential computers and cost too much to partition, parallel graph partitioning tools have been developed [1], [2], but their outcome is mixed. In particular, in the context of parallel graph ordering which is the one of this paper, they do not scale well, as partitioning quality tends to decrease, and thus fill-in tends to increase much, when the number of processors which run the program increase.

Consequently, graph ordering is the first target application of the PT-Scotch ("*Parallel Threaded* Scotch") software, a parallel extension of the sequential Scotch graph partitioning and ordering tool that we are currently developing within the ScAlApplix project. Graph ordering is a critical problem for the efficient factorization of symmetric sparse matrices, not only to reduce fill-in and factorization cost, but also to increase concurrency in the elimination tree, which is essential in order to achieve high performance when solving these linear systems on parallel architectures. We outline in this extended abstract the algorithms which we have implemented in PT-Scotch to parallelize the Nested Dissection ordering method [3].

II. Algorithms for efficient parallel reordering

The parallel computation of orderings in PT-Scotch uses three levels of concurrency. The first level is the implementation of the nested dissection method itself, which is straightforward thanks to the intrinsically concurrent nature of the algorithm. Starting from the initial graph, arbitrarily distributed across p processors, the algorithm proceeds as follows: once a separator has been computed in parallel, by means of a method described below, each of the p processors participates in the building of the distributed induced subgraph corresponding to the first separated part. This subgraph is then folded on the first $\lceil \frac{p}{2} \rceil$ processors, such that the average number of vertices per processor, which guarantees efficiency as it allows the shadowing of communications by a subsequent amount of computation, remains constant. The same procedure is used to build, on the $\lfloor \frac{p}{2} \rfloor$ remaining processors, the folded induced subgraph corresponding to the second part. These two constructions being completely independent, each of the computations of an induced subgraph and of its folding can be done in parallel, thanks to the temporary creation of an extra thread per processor. At the end of the folding process, the nested dissection process can recursively proceed independently on each subgroup of $\frac{p}{2}$ processors, until each of the subgroups is reduced to a single processor. From then on, the nested dissection process will go on sequentially, using the nested dissection routines of the Scotch library.

The second level of concurrency regards the computation of separators. The approach we have chosen is the now classical multi-level one [4]. It consists in repeatedly computing a set of increasingly coarser versions of the graph to separate, by finding vertex matchings which collapse vertices and edges, until the coarsest graph obtained is no larger than a few hundreds of vertices, then computing a separator on this coarsest graph, and projecting back this separator, from coarser to finer graphs, up to the original graph. Most often, a local optimization algorithm, such as Fiduccia-Mattheyses (FM) [5], is used in the uncoarsening phase to refine the partition that is projected back at every level, such that the granularity of the solution is the one of the original graph and not the one of the coarsest graph. The matching of vertices is performed in parallel by means of an asynchronous probabilistic multi-threaded algorithm. At the end of each coarsening step, the coarser graph is folded onto half of the processors that held the finer graph, in order to keep a constant number of vertices per processors, but it is also duplicated on the other half of the processors too. Therefore, the coarsening process can recursively proceed independently on each of the two halves, which results in an improvement of the quality of the separators, as only the best separator produced by the two halves is kept at the upper level.

The third level of concurrency regards the refinement heuristic which is used to improve the separators. FM-like algorithms do not parallelize well, as they are intrinsically sequential, and attempts to relax this strong sequential constraint can lead to severe loss of partition quality when the number of processors increase [1]. We have proposed and successfully tested in [6] a solution to this problem: since every refinement is performed by means of a local algorithm, which perturbs only in a limited way the position of the projected separator, the local refinement algorithm

LaBRI and INRIA Futurs
Université Bordeaux I
351, cours de la Libération, 33405 TALENCE, FRANCE
{cchevali|pelegrin}@labri.fr

Graph	Size ($\times 10^3$)		Average degree
	V	E	
audikw1	944	38354	81.28
conesphere1m	1055	8023	15.21
coupole8000	1768	41657	47.12
thread	29	2220	149.32

TABLE I
SOME OF OUR TEST GRAPHS.

Test case		Number of processes		
		2	16	128
audikw1	S	5.59e+12	5.32e+12	5.47e+12
	P	5.98e+12	7.42e+12	1.52e+13
conesphere1m	S	1.86e+12	1.86e+12	1.94e+12
	P	2.14e+12	3.05e+12	3.48e+12
coupole8000	S	7.44e+10	7.41e+10	7.41e+10
	P	8.14e+10	8.21e+10	9.19e+10
thread	S	3.70e+10	4.32e+10	4.65e+10
	P	4.38e+10	1.12e+11	–

TABLE II
CHOLESKY OPERATION COUNT (OPC) FOR PT-SCOTCH (S, TOP LINES) AND PARMETIS (P, BOTTOM LINES).

needs only to be passed a subgraph that contains the vertices that are very close to the projected separator. We have experimented that, when performing the refinement algorithm on band graphs that contains only the vertices that are at distance at most 3 from the projected separators, the quality of the finest separator is not significantly altered, and can even be improved in some cases. The advantage of constrained band FM is that band graphs are of a much smaller size than their parent graphs, and can therefore be used to run algorithms that would else be too costly to consider, such as evolutionary algorithms. What we have implemented is a multi-sequential approach: at every distributed uncoarsening step, a distributed band graph is created by using the very same algorithms as the one used to build each of the two separated subgraphs in the nested dissection process. Centralized copies of this band graph are then created on every participating processor. These copies can be used collectively to run a scalable parallel multi-deme genetic optimization algorithm [6], or fully independent runs of a full-featured sequential FM algorithm. The best refined band separator is projected back to the distributed graph, and the uncoarsening process goes on. Centralizing band graphs is an acceptable solution because for most graphs the size of the separators is of several orders of magnitude smaller that the size of the separated graphs: it is for instance in $O(n^{\frac{1}{2}})$ for 2D meshes, and in $O(n^{\frac{2}{3}})$ for 3D meshes [7].

III. EXPERIMENTAL RESULTS

PT-SCOTCH is written in ANSI C, with calls to the POSIX thread and MPI APIs. Some of the graphs that we used to run our tests are shown in Table I.

Table II presents the operation count of Cholesky factorization (OPC) yielded by the orderings computed with PT-SCOTCH and PARMETIS. The improvement in quality yielded by PT-SCOTCH is clearly evidenced, and increases along with the number of processes, as our local optimization scheme is not sensitive to the number of processes.

IV. CONCLUSION

We have outlined in this paper the parallel algorithms that we have implemented in PT-SCOTCH to compute in parallel efficient orderings of large graphs. The first results are encouraging, as they meet the expected performance requirements in term of quality, but have to be improved in term of scalability, because the current version of our asynchronous matching algorithm does too many communications that are not shadowed by computations. A multi-buffer version of the matching algorithm is therefore being developed.

Although it corresponds to a current need within the SCALAPPLIX project, to obtain as quickly as possible high quality orderings of graphs with a size of a few tens of millions of vertices, sparse matrix ordering is not the application field in which we expect to find the largest problem graphs, as existing parallel direct sparse linear system solvers cannot currently handle full 3D meshes of a size larger than about fifty million unknowns.

Therefore, basing on the software building blocks that we have already written, we plan to extend the capabilities of PT-SCOTCH to compute k-ary edge partitions of large meshes for subdomain-based iterative methods, as well as static mappings of process graphs, as the SCOTCH library does sequentially.

REFERENCES

[1] "METIS: Family of multilevel partitioning algorithms," `http://glaros.dtc.umn.edu/gkhome/views/metis`.

[2] "JOSTLE: Graph partitioning software," `http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/`.

[3] A. George and J. W.-H. Liu, *Computer solution of large sparse positive definite systems*, Prentice Hall, 1981.

[4] S. T. Barnard and H. D. Simon, "A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems," *Concurrency: Practice and Experience*, vol. 6, no. 2, pp. 101–117, 1994.

[5] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proceedings of the 19th Design Automation Conference*. 1982, pp. 175–181, IEEE.

[6] C. Chevalier and F. Pellegrini, "Improvement of the efficiency of genetic algorithms for scalable parallel graph partitioning in a multi-level framework," `http://www.labri.fr/~pelegrin/papers/scotch_efficientga.pdf`, submitted to EuroPar'2006.

[7] R. J. Lipton, D. J. Rose, and R. E. Tarjan, "Generalized nested dissection," *SIAM Journal of Numerical Analysis*, vol. 16, no. 2, pp. 346–358, Apr. 1979.